A Novel Approach for Detection and Analysis of Malware for Smart Devices

PhD SYNOPSIS

Submitted for Partial Fulfillment of the Award of

Degree of

Doctor of Philosophy in Computer Science & Engineering

School of Information Technology & Computer Science

Lingaya's Vidyapeeth

Faridabad, Haryana 121002

Research Supervisor:

Dr. R.C. Tripathi

Professor

School of CSE

Lingaya's Vidyapeeth

Research Scholar:

Mohd Abdul Rahim Khan

18PH.CS06

2018-2021

Lingaya's Vidyapeeth

23 July, 2019

Contents

1	Area/ Specialization of Proposed Research Work	1
2	Title of Proposed Research Work	1
3	Introduction	1
4	Objective	2
5	Hypothesis	3
6	Literature Review	3
7	Research Methodology	6
8	Data Analysis	8
9	References	8

A Novel Approach for Detection and Analysis of Malware for Smart Devices

1 Area/ Specialization of Proposed Research Work

Computer Science & Engineering/Information Security/ System Security /Artificial Intelligence

2 Title of Proposed Research Work

A Novel Approach for Detection and Analysis of Malware for Smart Devices.

3 Introduction

In the present scenario, Smart devices become more rapidly growing appliances with powerful computing, sensing, networking, and connecting with cloud storage. Recently, the need for ubiquitous computing has fuelled the development of smart device applications that can be clearly observed by millions of apps and their download counts in Google's official and alternative major Google app stores, respectively (197 billion in 2017) (Team, 2017).

The proliferation of mobile apps has led to two distinct revenue models, one of which is controlled by legitimate businesses, such as app developers, and the other by malicious hackers. Two of the most common methods of generating revenue are advertising and the sale of various services, such as the sale of a zombie host, malware advertising, and personal data sales. As the most common method for hijacking innocent profits, repacked apps (attackers take an app that is both benign and genuine, inject malicious code into it, and repackage the app again for distribution) are the most frequently used method. Zhou and Jiang found that 86% of the malicious apps found on smartphones were repackaged apps. In one study, repacked apps had a (W. Zhou et al., 2013) percent share in advertising revenue and a 10 percent share in user base (Gibler et al., 2013).

One of the most widely used attack methods for creating malware for smart devices is repackaging (Qin et al., 2014). It is possible to automate the process of repackaging by using a variety of available software.

The steps involved in repackaging are

i. Go to an official app store and get a well-known app.
ii. Open the package and dismantle it

iii. Embed the malicious payloads
iv. Re-assemble and pack
v. It's also possible to sign the repackaged app with bogus information.
vi. Distribute to a specific location.

The availability of various reverse-engineering tools makes it very easy to repack an android application, and the self-signing feature further increases the chances of repacking. The Federal Trade Commission (FTC) has released an alert on fake mobile apps.

Because of the substantial financial rewards and widespread use of mobile devices, attackers were forced to develop new mobile malware or port existing desktop malware to mobile platforms. Examples of ported PC malware, such as SpyEye and Zeus, are Spitmo and Zitmo(*New GnatSpy Mobile Malware Family Discovered*, 2017). It is worth mentioning that apart from repacked app, there are many other attach methods which used Android apps to attack Android platform.

In today's cyber era, the benefits and dependency of businesses and users on Android application is very high and no more questionable. In such case the repacked Android application along with other malicious apps pose a critical threat and cause high risk to users and businesses in terms of financial and personal losses. Due to financial and brand loss, many businesses are interested in detecting such malicious applications which serve the main reason to work on malicious Android application detection. Detecting malicious Android application is an active research area and there are many active research works going in this area(Gurulian et al., 2018; Ishii et al., 2016; Junaid et al., 2018; Kohli, 2017; Li et al., 2019; Saracino et al., 2016; Wang et al., 2015; Z. Zhou et al., 2018). Many of these recent and earlier works are not suitable to address malicious application detection and had limitations which leads to propose new technique to detect malicious Android applications.

4 OBJECTIVE

i. To propose novel Approach to detect malicious in Android application.

ii. To improve the scanning time and detection accuracy of malicious Android application detection techniques.

iii. To address the issue of obfuscation attack on malicious Android application detection techniques.

iv. To customized the proposed and existing detection techniques to address specific problems such as repacked and piggybacked Smartphone-based attacks.

v. Analysis of prevention malicious attack in Smartphone devices.

5 HYPOTHESIS

Malicious Smartphone Application poses critical Security Threat to Smart Environment

6 LITERATURE REVIEW

In order to identify visually similar apps, DroidEagle uses the layout resources within an app (Sun et al., 2015). App repositories are scanned, and applications on the host machine are detected, using the DroidEagle's (Linares-Vásquez et al., 2016)and HostEagle systems.

Using normalised opcode sequences and app permission requests, the authors of a recent work have proposed(Ali-Gombe et al., 2015) OpSeq, which calculates similarity scores. An Android application's unique fingerprint is created by combining structural and behavioural features, according to the authors. Signature-based detection works against obfuscation, but it only works against known malware.

Information Retrieval techniques and five semantic anchors: identifier, Android API, intent, permission and sensor are used in CLANdroid to identify similar apps. CLANdroid (Linares-Vásquez et al., n.d.) focuses primarily on finding similar apps that don't need to be repacked, such as searching for similar apps for a car booking service and repacking is distributing the same game apps by changing the developer's details or replacing the advertisement channel code, for example.

Authors have proposed a method for measuring app similarity based on claims of behaviour in one of their studies (Linares-Vásquez et al., 2016). Ontological analysis was used to augment the information retrieval method to extract raw features, which were then used as attributes to describe apps. A method known as agglomerative hierarchical clustering was employed to group the apps together(Mcilroy et al., 2017). More than 17,000 apps from the BlackBerry and Google app stores were used in experiments. From 0.02 to 0.41 and 0.03 to 0.21, respectively, the proposed method improves the existing categorization quality of Blackberry and Google stores respectively.

In order to locate and detect repacked applications, DroidMOSS uses fuzzy hashing to measure similarity(W. Zhou et al., 2012). A systematic investigation of six popular third-party Android app marketplaces found that between 5% and 13% of the apps hosted on these

marketplaces were repackaged for various purposes, including stealing or redirecting advertising revenue and introducing malware.

To detect repacked applications, the CTPH (Context Triggered Piece-wise Hash)(Hudel & Shehab, 2013) is used twice (Twice CTPH). The hash similarity calculation algorithm is also optimised by the authors, which improves the process's efficiency.

Crussel and colleagues Program decency graph (PDG) design by AnDarwin(Crussell et al., 2013) uses data dependency in the apps, data dependency is more typical to obscure than the semantic vector, it locates similar apps in storage. It's unnecessary to compare apps from different clusters.

(Yang et al., 2014)Yang et al. found the malicious payload in APKLancet. Because APKLancet does not maintain a signature database, it makes the implausible assumption that malicious payloads are completely separate from APKs. This is simply not true. Apps will produce a different outcome when sharing resources.

According to Zhou et al. (W. Zhou et al., 2013), if an app does not have a watermark, it appears to be a repackaged app. They point out that it is not easy to embed the watermarks in a java code. Non-trivial data structure is converted into a watermark value. Once the graph has been decoded, the corresponding value can be found and compared to the developer's watermark.

Detecting whether an app has been repackaged has been proposed by Vidas et al.(Vidas & Christin, 2013), but distributing the original public and private keys is a challenge. Changing the package name and providing a web address for his own public key will cause this mechanism to fail.

Table 1 Reviewed and synthesised relevant research on Android detection methods

(Y. Zhou et al., 2012)Works	**Techniques**	**Observations**
DroidEagle(Li et al., 2019)	Layout resources tree	Static method, Tree comparisons is costly
OpSeq (Ali-Gombe et al., 2015)	Sequence of opcode and permissions requests	Static method, work only for known malware
DroidMOSS (Gurulian et al., 2018)	Fuzzy hashing	Cannot handle Obfuscation

AnDrawin (Crussell et al., 2013)	Min Hashing Technique	Even without comparing each app pair, it is possible to make a determination. Obfuscation is much more difficult when there are data dependencies involved.
AppInk(W. Zhou et al., 2013)	Watermarking	Android apps can now include a watermark. The watermark is bogus and should be avoided.
APKLancet(Yang et al., 2014)	Signature based	Get to know what's bad in the payload itself.
AppIntegrity(Vidas & Christin, 2013)	Public and Private Key	Distributing the original public key that corresponds to the app's private key is a challenge.
DroidRanger (Y. Zhou et al., 2012)	Permission-based behavioral fingerprinting	Benign apps often request a combination of permissions, so you can tell if there is any potentially harmful behaviour present by looking for this pattern.
RiskRanker (Wang et al., 2015)	Systematic approach	Using native code, dynamic class loading, and callback handlers, an application can discover if dangerous behaviour is present.
CrowDroid (Burguera et al., 2011)	Application based on logs collected	Only Linux-specific data, such as open files, IPC, and Android-specific data, can be extracted

Identification of Research Gap and Problem

The initial literature review of the proposed research domain indicates that earlier works has limitations in term of scanning time, accuracy and obfuscation and also the main focus of few of earlier works are not to detect malicious application but other similar tasks such as classification of smart applications in categories, data leaking detection (Garcia et al., 2018; Lu et al., 2013)etc. These limitations sought novel techniques to address the proposed research problem and work effectively in detection of malicious in Smartphone apps.

Expected Impact on Academics/Industry

The impact of proposed research on academics and industry can be described as follows:

- Detection, Removal and Prevention of malicious app will make the Smartphone app environment safer for users.

- A safer Smartphone Apps environment will increase the user's participation and that will help to grow smartphone marketplace and will help to generate revenue.
- With the help of proposed detection techniques, the smartphone apps marketplaces will be able to keep malicious smartphone apps out of their platform.

7 RESEARCH METHODOLOGY

An experiment based research methodology is adopted that is the proposed technique will be implemented in term of computer programs and will be tested with various sample cases. The experiments will be carried out in a controlled environment and primary and secondary dataset will be used as per the requirement of the experiment. Figure 1 visually represent the experimental setup and the process.

The proposed method for detecting repacked Android applications is ImageHash, also known as perceptual hashing. A one-to-many or many-to-many hash comparison is an O(1) string comparison problem, but searching with a hash value is much faster and has a time complexity of only O(1).For our hash comparison, we used a tree-based searching method. Figure 1 depicts the proposed architecture's block diagram. Detailed explanations of the proposed architecture's various components are also provided.

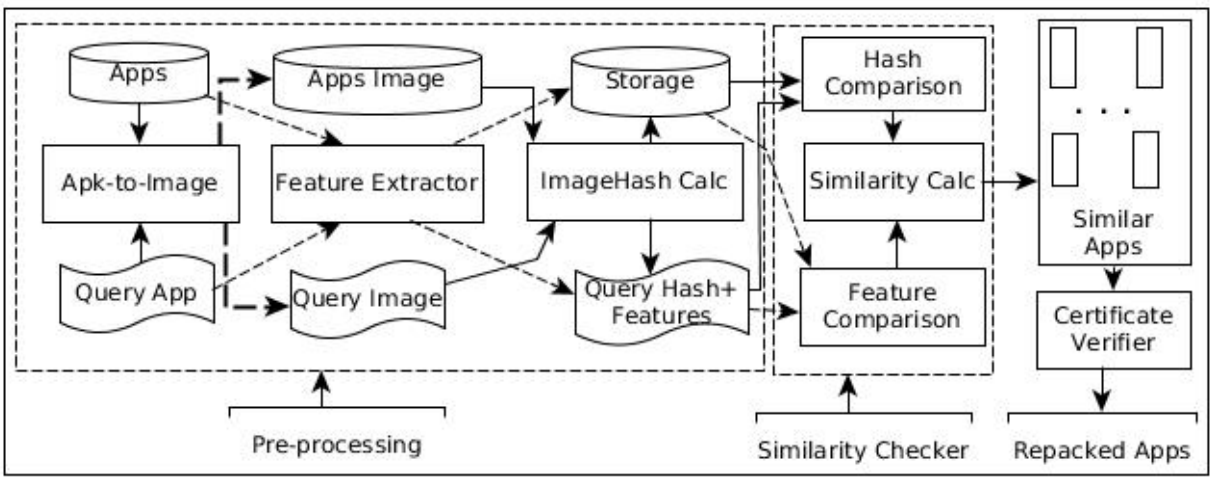

Figure 1: Proposed Work Architecture

7.1 Apk to Image conversion

When storing any digital file, the memory device stores it as a stream of bits, each of which contains one or more zeroes or ones. Each apk file is read as a binary stream and then each eight-bit group is stored in its own image file. The grayscale image's pixel values range from 0 to 255, which can be represented by a group of eight bits.

To begin the process of generating an image hash, an apk must first be converted to a grayscale image using the method described above. In our experiment, we set the width of the image to 1024 and the length to vary according to the size of the apk. It takes the target and query apps as input, and converts them all to grayscale image files, which can then be used by other parts of the application.

The binary content of the executable is first numerically represented as a discrete one-

dimensional signal in the range [0-255] in the above-mentioned image conversion technique, and then it is transformed into a two-dimensional grayscale image (Nataraj et al., 2013).

7.2 Feature Extraction

Image-hash is extremely accurate in many applications, such as image-based content search. We used static app features as a filter for this project and calculated similarity based on those features. The proposed work's feature extractor (Ref. Figure 1) component extracts the static features of these apps. An image hash and pre-defined features can be extracted from a target and a query app using feature extractor components. Selected static features include the total number of API calls as well as the total number of classes and methods as well as file size, request permissions, and layout strings. All of the features listed above can be static, which means they can be extracted without having to run any programmes. You can use these static features to sort through similar apps and find the best one out of the bunch.

7.3 Image hashing

For the purposes of comparing the initial data, perceptual hash algorithms produce hashes. If the generated hashes (in a certain space with certain measures) are more similar to each other, this indicates how similar the initial data is to that hashed data.

As a result, we gain the following benefits:

Storage efficiency (saving hashes is cheaper than keeping the initial images)

Authors can organise hashes into different structures like KD-Tree, MVP-Tree, and others and perform the operations with complexity O(N), a linear one, to save computing power.

7.4 Hash comparison

As you can imagine, it is extremely time-consuming, technically difficult, and in general does not make any sense to compare images pixel-by-pixel. O(N^2) is the metric used to estimate the complexity of this comparison. To begin, scan and calculate the fingerprints of N images. We can find all duplication in O(N) time if we consider 100% similarity as a duplicate (which implies that all duplications have identical fingerprints). It takes O(N N) time to find all instances of duplication when you use a similarity threshold of 90%.

7.5 Hash distance Calculation

Distances between two hashes can be calculated using the same formula used for searching for similar terms and other text-based tasks: hashing. There are a number of well-known and effective hash distance calculation methods based on these string distance computations. Others will be used in the proposed study if the methodology proves to be effective.

8 DATA ANALYSIS

To verify the effeteness and correctness of the proposed solutions we need to perform various statistical tests which will help to prove our hypothesis. For data analysis we need data points, in Android detection domain each class of Android applications acts as data point, for example for malicious and benign classification, set of malicious and benign Smartphone applications are data points. As during this research work focus is to proposed various detection methods based on Machine Learning (ML) techniques or related to similar domains. So during data analysis various performance metrics will be used from the ML domain such as accuracy, False Positive Rate (FPR), True Positive Rate(TPR), Area Under Curve (AUC) etc. During data analysis phase, these metrics will help to compare the proposed technique with earlier works presented in the literature.

9 REFERENCES

Ali-Gombe, A., Ahmed, I., Richard III, G. G., & Roussev, V. (2015). Opseq: Android malware fingerprinting. *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, 1–12.

Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). Crowdroid: Behavior-based malware detection system for android. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 15–26.

Crussell, J., Gibler, C., & Chen, H. (2013). Andarwin: Scalable detection of semantically similar android applications. *European Symposium on Research in Computer Security*, 182–199.

Garcia, J., Hammad, M., & Malek, S. (2018). Lightweight, obfuscation-resilient detection and family identification of android malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *26*(3), 1–29.

Gibler, C., Stevens, R., Crussell, J., Chen, H., Zang, H., & Choi, H. (2013). Adrob: Examining the landscape and impact of android application plagiarism. *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, 431–444.

Gurulian, I., Markantonakis, K., Cavallaro, L., & Mayes, K. (2018). Reprint of "You can't touch this: Consumer-centric android application repackaging detection." *Future Generation Computer Systems*, *80*, 537–545.

Hudel, C., & Shehab, M. (2013). Optimizing search for malware by hashing smaller amounts of data. *World Congress on Internet Security (WorldCIS-2013)*, 112–117.

Ishii, Y., Watanabe, T., Akiyama, M., & Mori, T. (2016). Clone or Relative? Understanding the Origins of Similar Android Apps. *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, 25–32.

Junaid, M., Ming, J., & Kung, D. (2018). Statedroid: Stateful detection of stealthy attacks in android apps via horn-clause verification. *Proceedings of the 34th Annual Computer Security Applications Conference*, 198–209.

Kohli, A. (2017). Decisiondroid: A supervised learning-based system to identify cloned android applications. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 1059–1061.

Li, L., Bissyandé, T. F., & Klein, J. (2019). Rebooting research on detecting repackaged android apps: Literature review and benchmark. *IEEE Transactions on Software Engineering*, *47*(4), 676–693.

Linares-Vásquez, M., Holtzhauer, A., & Poshyvanyk, D. (n.d.). *On automatically detecting similar Android apps (CLANdroid)*.

Linares-Vásquez, M., Holtzhauer, A., & Poshyvanyk, D. (2016). On automatically detecting similar Android apps. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 1–10.

Lu, X., Zhuge, J., Wang, R., Cao, Y., & Chen, Y. (2013). De-obfuscation and detection of malicious PDF files with high accuracy. *2013 46th Hawaii International Conference on System Sciences*, 4890–4899.

Mcilroy, S., Shang, W., Ali, N., & Hassan, A. E. (2017). User reviews of top mobile apps in Apple and Google app stores. *Communications of the ACM*, *60*(11), 62–67.

Nataraj, L., Kirat, D., Manjunath, B. S., & Vigna, G. (2013). Sarvam: Search and retrieval of malware. *Proceedings of the Annual Computer Security Conference (ACSAC) Worshop on Next Generation Malware Attacks and Defense (NGMAD)*.

New GnatSpy Mobile Malware Family Discovered. (2017, December 18). Trend Micro. https://www.trendmicro.com/en_us/research/17/l/new-gnatspy-mobile-malware-family-discovered.html

Qin, Z., Zhang, X., Zhang, Q., & Yang, Z. (2014). An efficient method of detecting repackaged android applications. *International Conference on Cyberspace Technology (CCT 2014)*, 1–4.

Saracino, A., Sgandurra, D., Dini, G., & Martinelli, F. (2016). Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, *15*(1), 83–97.

Sun, M., Li, M., & Lui, J. C. (2015). DroidEagle: Seamless detection of visually similar Android apps. *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 1–12.

Team, C. P. M. R. (2017). *The Judy Malware: Possibly the largest malware campaign found on Google Play*. Retrieved from checkpoint. com: http://blog. checkpoint. com/2017/05/25/judy ….

Vidas, T., & Christin, N. (2013). Sweetening android lemon markets: Measuring and combating malware in application marketplaces. *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, 197–208.

Wang, H., Guo, Y., Ma, Z., & Chen, X. (2015). Wukong: A scalable and accurate two-phase approach to android app clone detection. *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 71–82.

Yang, W., Li, J., Zhang, Y., Li, Y., Shu, J., & Gu, D. (2014). APKLancet: Tumor payload diagnosis and purification for android applications. *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, 483–494.

Zhou, W., Zhang, X., & Jiang, X. (2013). AppInk: Watermarking android apps for repackaging deterrence. *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 1–12.

Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, 317–326.

Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012). Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. *NDSS*, *25*(4), 50–52.

Zhou, Z., Sun, C., Lu, J., & Lv, F. (2018). Research and implementation of mobile application security detection combining static and dynamic. *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, 243–247.